Journal of Discrete Algorithms 7 (2009) 239–248

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Algorithms for optimal outlier removal ☆

Rossen Atanassov, Prosenjit Bose, Mathieu Couture, Anil Maheshwari, Pat Morin *, Michel Paquette, Michiel Smid, Stefanie Wuhrer

School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Canada K1S 5B6

ARTICLE INFO

Article history:
Received 7 August 2006
Received in revised form 9 August 2007
Accepted 19 December 2008
Available online 14 January 2009

Keywords:
Computational geometry
Outlier removal
Computational statistics

ABSTRACT

We consider the problem of removing c points from a set S of n points so that the remaining point set is optimal in some sense. Definitions of optimality we consider include having minimum diameter, having minimum area (perimeter) bounding box, having minimum area (perimeter) convex hull. For constant values of c, all our algorithms run in $O(n \log n)$ time.

1. Introduction

Motivated by the problem of removing outliers in a data set, this paper considers the following problem: Let S be a set of n points in $\mathbb{R}^2$ and let $f : 2^{\mathbb{R}^2} \mapsto \mathbb{R}$ be a function mapping point sets onto real values. We consider the problem of selecting a subset $S' \subseteq S$, $|S'| = c$ such that $f(S \setminus S')$ is minimum. The objective functions f that we consider are:

(1) $f(X)$ is the diameter of X,
(2) $f(X)$ is the area/perimeter of the smallest axes-parallel rectangle containing X, and
(3) $f(X)$ is the area/perimeter of the convex hull of X.

We call these problems the *f-based c outlier removal problems*. We are particularly interested in the case when c (the number of outliers) is small. The functions we consider all have the property that their values can become very large in the presence of even one outlier. Indeed, the same types of outliers that can skew non-robust summary statistics such as the mean and standard deviation make these functions very large. The choice of which function to use for outlier removal depends on the particular application. Methods based on bounding boxes are well-suited for situations where the data consists of two measured variables in which scale is important. The methods based on diameter are well-suited to situations where rotations of the data are irrelevant. The methods based on area of the convex hull are well-suited to situations where even affine transformations of the data are irrelevant.

☆ This work was partly supported by NSERC.
* Corresponding author.
E-mail address: morin@scs.carleton.ca (P. Morin).

1570-8667/$ – see front matter
doi:10.1016/j.jda.2008.12.002

1.1. Previous work

The outlier removal problems stated above and similar problems are fairly well-studied problems in the field of computational geometry. However, most work thus far has focused on the case when c is large.[1] More specifically, most research has been on the problem of finding a k point subset (a k-cluster) $X \subseteq S$, $|X| = k$, such that $f(X)$ is minimum. These are the same problems studied in the current paper with $k = n - c$ except that our focus is on large values of k (small c) and most existing research focuses on designing efficient algorithms for small values of k.

Minimum diameter. The problem of finding a subset of S of size k with minimum diameter has been studied by Aggarwal et al. [1] who give an $O(k^{2.5}n\log k + n\log n)$ time algorithm. Using a different approach, Eppstein and Erickson [12] improve the running time to $O(n\log n + k^2 n\log^2 k)$.

Minimum-perimeter/area enclosing rectangle. Aggarwal et al. [1] also give an $O(k^2 n\log n)$ time algorithm to find the subset of S with the minimum-perimeter enclosing axes-parallel rectangle. Eppstein and Erickson [12] improve the running time to $O(n\log n + k^2 n)$. Segal and Kedem [19] present an algorithm for this problem that runs in $O(n + k(n-k)^2)$ time, provided $\frac{n}{2} < k \leqslant n$. Their algorithm can also find the minimum area rectangle.

Minimum-perimeter/area convex hull. The problem of finding a subset of S of size k that has the least perimeter convex hull was first considered over 20 years ago by Dobkin et al. [8] who gave an $O(k^2 n\log n + k^5 n)$ time algorithm. This algorithm can be improved to run in $O(k^2 n\log n + k^4 n)$ time using techniques of Aggarwal et al. [1]. Eppstein and Erickson [12] hold the record with a running time of $O(n\log n + k^3 n)$.

The problem of finding a subset of S of size k that has the minimum area convex hull was considered by Eppstein et al. [13] and later by Eppstein [11] who give $O(kn^3)$ and $O(n^2\log n + k^3 n^2)$ time algorithms for this problem, respectively.

1.2. New results

For fixed values of k, the results described above give $O(n\log n)$ or $O(n^2\log n)$ time algorithms for finding the k point subset of S that minimizes f. However, when k is close to n, all but one of the algorithms cited above require $\Omega(n^3)$ time. The one exception to this is the algorithm of Kedem and Segal [19], which runs in $O(n + k(n-k)^2) = O(n + (n-c)c^2)$ and is therefore fast when k is very large.

In the current paper we consider specifically the case when $k = n - c$. For problem 1 (diameter) we obtain an $O(n\log n + cn + c^6\log^2 c)$ time algorithm. For problem 2 (minimum area/perimeter bounding rectangle) we obtain an $O(n + c^3)$ time algorithm. For problem 3 (minimum area/perimeter convex hull) we obtain an $O(n\log n + \binom{4c}{2c}(3c)^{c+1}n)$ time algorithm. We also give $\Omega(n\log n)$ lower bounds for problems 1 and 3, thereby showing that, for constant values of c, our algorithms are optimal.

We obtain many of our results by borrowing techniques from the theory of fixed-parameter tractability (FPT) that are typically applied to NP-hard problems [9]. In particular, our solutions to problems 1 and 2 use the technique of *kernelization* by using a fast algorithm to find a problem kernel of size $c^{O(1)}$ whose solution gives a solution to the original problem. Our solution to problem 3 can also be viewed as a form of the bounded search tree method [9], although its implementation differs considerably from that used in typical FPT algorithms.

The remainder of the paper is organized as follows: Section 2 presents an algorithm for the diameter-based c outlier removal problem. Section 3 presents algorithms for the (bounding box) area/perimeter-based c outlier removal problems. Section 4 presents algorithms for (convex hull) area/perimeter-based c outlier removal problems. An abstract of the results in Section 4 has appeared in the proceedings of the 18th Canadian Conference on Computational Geometry [2].

2. Minimizing diameter

The *diameter* of a set X is defined as $\max\{\|pq\|\colon\ p, q \in X\}$. In this section we consider the *diameter-based c outlier removal problem* of finding a subset $S' \subseteq S$, $|S'| = c$ such that the diameter of $S \setminus S'$ is minimum over all choices of S'. We begin by showing that this problem has an $\Omega(n\log n)$ lower bound even for the case $c = 1$.

2.1. The lower bound

Theorem 1. *In the algebraic decision tree model of computation the diameter-based* 1 *outlier removal problem has an* $\Omega(n\log n)$ *lower bound.*

Proof. Computing the diameter of a set S of n points in $\mathbb{R}^2$ has an $\Omega(n\log n)$ lower bound in the algebraic decision tree model [17]. We simply observe that the point x that defines the optimal solution is one of the two points that define the

[1] One notable exception to this statement is the work on linear programming with c violations, which includes the problem of finding a subset S' of S, $|S'| = c$ such that the radius of the smallest disk containing $S \setminus S'$ is minimum. For this problem, Matoušek gives an $O(n\log n + c^3 n^{\epsilon})$ time algorithm [16] and Chan [4] gives an $O(n\beta(n)\log n + c^2 n^{\epsilon})$ time algorithm, where $\epsilon > 0$ is an arbitrarily small constant and $\beta(\cdot)$ is related to the inverse Ackermann function.

diameter of S. Thus, given x we can compute the diameter of S in $O(n)$ time by computing the distance from x to every other point of S. It must therefore take $\Omega(n \log n)$ time to determine x. □

2.2. The algorithm

Next we give an algorithm for the diameter-based c outlier removal problem. We do this by first considering a more general problem in $\mathbb{R}^{n^2}$ and then use the resulting algorithm to find a set $\tilde{S}$ of $O(c^2)$ points with the property that any optimal set S' for S is also an optimal set for $\tilde{S}$.

Let A denote an $n \times n$ symmetric matrix of non-negative real numbers whose diagonal entries are all equal to 0 and whose off-diagonal entries are all unique. An example of such a matrix, which is used for diameter-based outlier removal, is the matrix in which $A_{i,j}$ denotes the distance from point i to point j in an n point set in which all interpoint distances are unique.

Let $A_{i,\star}$ and $A_{\star,i}$ denote the ith row and column of A, respectively. For a set I of indices we denote by $A \setminus I$ the submatrix of A obtained by deleting $A_{i,\star}$ and $A_{\star,i}$ for every $i \in I$ from A. We consider the problem of selecting a set T of c indices, such that the largest value in $A \setminus T$ is minimum over all choices of T. We begin by showing that the algorithm DeepSet described below selects a set $\tilde{T}$ of at most $(c+1)^2$ indices that are a superset of some optimal set T.

In the following, *marking* an index i means adding i to $\tilde{T}$ and *deleting* an index i means removing row i and column i from A. In words, Algorithm DeepSet(A, c) finds the maximum entry $A_{i,j}$ of A, marks the indices of the $c+1$ largest entries in $A_{i,\star}$ and $A_{\star,j}$, and finally deletes i and j. The algorithm repeats this process $c+1$ times. In pseudocode, Algorithm DeepSet(A, c) is given below:

DeepSet(A, c)
1: **for** $g = 1$ to $c+1$ **do**
2: $A_{i,j} \leftarrow$ maximum entry of A
3: $x \leftarrow (c+1)$-st largest entry in $A_{i,\star}$
4: $\tilde{T} \leftarrow \tilde{T} \cup \{k\colon A_{i,k} \geqslant x\}$ {* mark $c+1$ largest entries in row/column i *}
5: $x \leftarrow (c+1)$-st largest entry in $A_{\star,j}$
6: $\tilde{T} \leftarrow \tilde{T} \cup \{k\colon A_{k,j} \geqslant x\}$ {* mark $c+1$ largest entries in row/column j *}
7: $A \leftarrow A \setminus \{i, j\}$ {* delete i and j *}
8: **return** $\tilde{T}$

Lemma 2. *Let T be a set of c indices such that the largest value in $A \setminus T$ is minimum over all choices of T. Then Algorithm* DeepSet(A, c) *returns a superset $\tilde{T} \supseteq T$.*

Proof. To prove the lemma we first impose an order on the elements of $T = \{t_1, \ldots, t_c\}$, such that the maximum element of $A \setminus \{t_1, \ldots, t_{i-1}\}$ is in $A_{t_i,\star}$ (and symmetrically in $A_{\star,t_i}$). This ordering always exists, otherwise T is not optimal. Let $T_i = \{t_1, \ldots, t_i\}$ and let $\tilde{T}_i$ denote the set of indices contained in $\tilde{T}$ after the execution of the ith iteration of the algorithm DeepSet. Let D_i denote the set of $2i$ indices that have been deleted after the ith iteration of the algorithm DeepSet. For convenience we use the convention $T_0 = \tilde{T}_0 = D_0 = \emptyset$. The following claim is easily established by induction on i:

Claim 1. *The maximum value in $A \setminus D_i$ is less than or equal to the maximum value in $A \setminus T_i$.*

Next we prove, by induction on i, that $\tilde{T}_i \supseteq T_i$. The base case, $i = 0$, is trivial. For the inductive step, we show that $\tilde{T}_i \supseteq T_i$ if $\tilde{T}_{i-1} \supseteq T_{i-1}$. Let $A_{k,\ell}$ be the maximum element of $A \setminus T_{i-1}$. Recall that, by the ordering we have chosen for T, this implies that $T_i = T_{i-1} \cup \{t_i\}$ where $t_i \in \{k, \ell\}$. We will show that both k and ℓ are in $\tilde{T}_i$. Three cases are possible:

(1) $k \notin \tilde{T}_{i-1}$ and $\ell \notin \tilde{T}_{i-1}$. Since the marked elements are a superset of the deleted elements, $A_{k,\ell}$ is an element of $A \setminus D_{i-1}$. Therefore, by Claim 1, $A_{k,\ell}$ is the largest element in $A \setminus D_{i-1}$. Thus, k and ℓ are marked (and deleted) during the ith iteration of DeepSet so $\{k, \ell\} \subseteq \tilde{T}_i$.
(2) $k \in \tilde{T}_{i-1}$ and $\ell \in \tilde{T}_{i-1}$. Then $\{k, \ell\} \subseteq \tilde{T}_{i-1} \subseteq \tilde{T}_i$.
(3) Without loss of generality $k \in \tilde{T}_{i-1}$ and $\ell \notin \tilde{T}_{i-1}$. Two cases are possible.
   (a) k was marked and deleted during the first $i-1$ iterations of algorithm DeepSet. Hence, the indices of the $c+1$ largest entries in $A_{k,\star}$ are in $\tilde{T}_{i-1}$ and none of these indices is ℓ. On the other hand, T_{i-1} contains only $i-1 < c+1$ indices and none of these is k. Thus, it must be that $A \setminus T_{i-1}$ contains a value $A_{k,\ell'} > A_{k,\ell}$, but this contradicts the ordering of T.
   (b) The index k was marked, but not deleted, during the first $i-1$ iterations of algorithm DeepSet. Thus, the same argument used in Case 1 above implies that $\{k, \ell\} \subseteq \tilde{T}_i$.

This completes the proof of Lemma 2. □

Carefully inspecting the proof of Lemma 2 shows that it actually implies the following slightly stronger statement (because $\tilde{T}_{c+1}$ contains the indices of the largest element in $A \setminus T_c$).

Lemma 3. *Let $\tilde{A}$ be the submatrix of A induced by the row and column indices in the set $\tilde{T}$ produced by Algorithm* DeepSet. *Then the value of the optimal solution for $\tilde{A}$ is equal to the value of the optimal solution for A.*

We can use the algorithm DeepSet to reduce a problem of size n to one of size $O(c^2)$. The set S implicitly defines an $n \times n$ *distance matrix* A where $A_{i,j}$ is the distance from the ith point in S to the jth point in S. Note that A is a symmetric matrix of non-negative real numbers with zero values along the diagonal. To ensure that A also has the second property we perform comparisons between the elements of A lexicographically using the key $(A_{i,j}, i, j)$ for the element $A_{i,j}$. In this way we can run algorithm DeepSet on A to obtain a superset of the indices of points that need to be removed to minimize the diameter of S.

Lemma 4. *There exists an $O(n \log n + cn + c^6 \log^2 c)$ time algorithm to solve the diameter-based c outlier removal problem.*

Proof. To execute algorithm DeepSet on the matrix A implicitly defined by S, we first preprocess S, in $O(n \log n)$ time, using the deletion-only convex hull structure of Hershberger and Suri [15], which allows the deletion of a point in $O(\log n)$ time and makes it possible to find the diameter in $O(n)$ time [20].

Denote the two points forming the diameter of S by p and q. After the preprocessing described above, p and q can be found, and deleted, in $O(n)$ time. Next, using an $O(n)$ time selection algorithm we mark the $c+1$ furthest points from p and q. Repeating this $c+1$ times we obtain, in $O(cn)$ time a set $\tilde{S}$ of $O(c^2)$ marked points such that the optimal solution S' for S is also the optimal solution for $\tilde{S}$.

Once we have computed $\tilde{S}$ we apply the algorithm of Eppstein and Erickson [12] to find the set S' in $O(c^2(c^2)^2 \log^2(c^2)) = O(c^6 \log^2 c)$ time. These three steps (preprocessing, finding $\tilde{S}$ and finding S') take a total time of $O(n \log n + cn + c^6 \log^2 c)$, as promised. □

Remark. The DeepSet algorithm can be extended to handle slightly more general problems for which the matrix A is not necessarily symmetric. The only modification needed for this extension is that the indices of the $c+1$ largest entries in $A_{i,\star}$ and in $A_{\star,i}$ as well as $A_{j,\star}$ and $A_{\star,j}$ should be marked. This modification at most doubles the size of the resulting set $\tilde{T}$ of marked indices.

3. Minimizing the enclosing rectangle

In this section we propose a simple algorithm for the following (*bounding box*) *area/perimeter-based c outlier removal problem*: given a set S of n points in the plane, find an axes-parallel closed rectangle R of minimum area/perimeter that has exactly c points of S in its exterior. Our algorithm first computes a set of $O(c)$ candidate points and then uses the algorithm of Kedem and Segal [19] to compute the desired rectangle. The main steps of our algorithm are given by the following pseudocode:

FindOptimalRectangle(S, c)

1: Find a subset $K \subseteq S$ consisting of c leftmost, c rightmost, c topmost and c bottommost points in S.
2: Compute the smallest axes-parallel bounding box B for points in $S \setminus K$, and then delete $S \setminus K$.
3: Insert c points at the bottom-left corner of B. Also insert c-points at the top-right corner of B. Let the set of newly inserted points be I.
4: Compute the minimum area/perimeter axes-parallel rectangle R that encloses exactly $c + |K|$ points out of $2c + |K|$ points in the set $K \cup I$ using the algorithm in [19].

Lemma 5. *The bounding box B is in the interior of the rectangle R reported by* FindOptimalRectangle.

Proof. The proof is by contradiction. Assume that the box B is not completely in the interior of R. Then at least one of the points in the set I is in the exterior of R. Let p be one such point. Imagine the coordinate system with its origin at p. Notice that R does not overlap with at least one of the following four halfspaces: (1) $y \geqslant 0$ (2) $x \geqslant 0$ (3) $y \leqslant 0$ (4) $x \leqslant 0$. Each of these half spaces contains more than c points (including p). Hence there are more than c points in the exterior of R, a contradiction. □

Theorem 6. *There exists an $O(n + c^3)$ time algorithm to solve the (bounding box) area/perimeter-based c outlier removal problem.*

Proof. Correctness follows from Lemma 5. As for the complexity analysis, Steps 1–3 take $O(n)$ time. Since $|K| \leqslant 4c$, Step 4 requires $O(c^3)$ time [19]. Hence the overall time complexity of the algorithm is $O(n + c^3)$. □

4. Minimizing the convex hull

In this section we consider the following (*convex hull*) *area/perimeter-based c outlier removal problems*: Given a set S of n points, find a subset $S' \subseteq S$, $|S'| = c$ such that the area/perimeter of the convex hull of $S \setminus S'$ is minimum over all choices of S'. Throughout this section, we use $\text{conv}(X)$ to denote the convex hull of the point set X. We start by giving an $\Omega(n \log n)$ lower bound, even for the case when $c = 1$.

4.1. The lower bound

Theorem 7. *In the algebraic decision tree model of computation, the* (*convex hull*) *area/perimeter-based* 1 *outlier removal problem has an* $\Omega(n \log n)$ *lower bound.*

Proof. The proof is by a reduction from the problem Set-Equality, of determining whether two sets A and B of real numbers are equal, which has an $\Omega(n \log n)$ lower bound in the algebraic decision tree model [3]. The Set-Equality problem can be mapped to an outlier removal problem on the point multiset $S = S_A \uplus S_B$ where $S_A = \{(x, x^2)\colon\ x \in A\}$, $S_B = \{(x, x^2)\colon\ x \in B\}$ and $\uplus$ denotes multiset union. The sets A and B are equal if and only if for every $p \in S$, $\text{conv}(S) = \text{conv}(S \setminus \{p\})$. □

4.2. The algorithm

In this subsection we present an algorithm to solve the (convex hull) area/perimeter-based c outlier removal problem. Throughout this subsection we will simply discuss the area-based problem. The bulk of the computational work in the algorithm is done by a dynamic programming subroutine that handles area or perimeter equally well. We start with some geometric preliminaries.

4.2.1. Preliminaries

The *convex layers* $S_0, \ldots, S_k$ of S are defined as follows: S_0 is the subset of S on the boundary of $\text{conv}(S)$. S_i, for $i \geqslant 1$ is the subset of S on the boundary of $\text{conv}(S \setminus \bigcup_{j=0}^{i-1} S_j)$. The convex layers of S can be computed in $O(n \log n)$ time [5,15] or, more simply, the first c convex layers can be computed in $O(cn \log n)$ time by repeated applications of any $O(n \log n)$ time convex hull algorithm. For the remainder of this paper we will use the notation $p_{i,j}$ to denote the (j mod $|S_i|$)th point of S_i, and use the convention that $p_{i,0}, \ldots, p_{i,|S_i|-1}$ occur in counterclockwise order on the boundary of $\text{conv}(S_i)$.

Once the first $c+1$ convex layers $S_0, \ldots, S_c$ have been computed, we can find, in $O(c^2 n)$ time, for each point $p_{i,j}$ on layer i and for each layer $i' > i$ and $i' \leqslant c$ the two points $p_{i',k}$ and $p_{i',\ell}$ such that the line through $p_{i,j}$ and $p_{i',k}$ (respectively $p_{i,j}$ and $p_{i',\ell}$) is tangent to $S_{i'}$. This is accomplished by a simple walk around S_i, updating tangents $p_{i',k}$ and $p_{i',\ell}$ as we proceed.

Consider a point $p_{0,j} \in S_0$ and refer to Fig. 1. If we remove $p_{0,j}$ from S then a (possibly empty) sequence $p_{1,k}, \ldots, p_{1,\ell}$ of S_1 appears on the boundary of $\text{conv}(S \setminus \{p_{0,j}\})$. When this happens we say that $p_{1,k}, \ldots, p_{1,\ell}$ is *exposed*. This exposed sequence can be obtained from the preprocessing described above by using two tangents joining $p_{0,j-1}$ and $p_{0,j+1}$ to $p_{1,k}$ and $p_{1,\ell}$, respectively. Finding the two tangent points $p_{1,k}$ and $p_{1,\ell}$ takes $O(1)$ time and traversing the sequence takes $O(t_j)$ time, where $t_j = \ell - k + 1$.

Once we have removed a point $p_{0,j}$ from S_0, if we know the area (or perimeter) of $\text{conv}(S)$, then we can compute the area (or perimeter) of $\text{conv}(S \setminus \{p_{0,j}\})$ in $O(t_j)$ time. We do this by computing the area of the triangle $\triangle p_{0,j-1} p_{0,j} p_{0,j+1}$ and subtracting from it the area of $\text{conv}(\{p_{0,j-1}, p_{0,j+1}\} \cup \{p_{1,k}, \ldots, p_{1,\ell}\})$. This gives us the difference in area (perimeter) between $\text{conv}(S)$ and $\text{conv}(S \setminus \{p_{0,j}\})$.

In this section we present our algorithm for solving the perimeter-based and area-based outlier removal problems. Our solution to both problems is to enumerate all the combinatorial types of solutions of size c. For each such solution type, we then use a combination of divide-and-conquer and dynamic programming to find the optimal solution of that particular solution type. Before presenting the general algorithm, it will be helpful to discuss the special cases $c = 1$ and $c = 2$ to illustrate the principles involved.

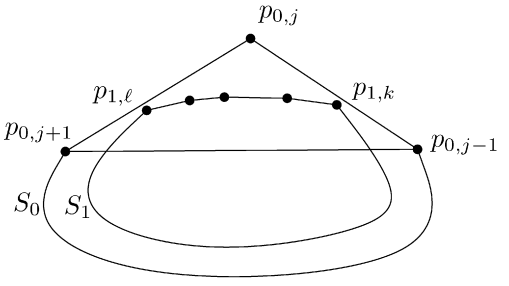

Fig. 1. Removing a point $p_{0,j}$ from S_0 exposes a chain $p_{1,k}, \ldots, p_{1,\ell}$ of S_1.

4.2.2. Removing 1 outlier

The case $c = 1$ asks us to remove 1 point of S so that the convex hull of the resulting set has minimum area. This can be solved as follows: We first compute the two convex layers S_0 and S_1 in $O(n \log n)$ time and preprocess them for the tangent queries described in the previous section. We then determine, for each point $p_{0,j} \in S_0$ the difference in area between $\text{conv}(S)$ and $\text{conv}(S \setminus \{p_{0,j}\})$ using the method described in the previous section. This process takes $O(1+t_j)$ time, where t_j is the number of vertices of S_1 exposed by the removal of $p_{0,j}$. We output the point $p_{0,j}$ that gives the largest difference in area.

To analyze the overall running time of this algorithm we observe that any particular point $p_{1,k} \in S_1$ appears in at most two triangles $\triangle p_{0,j-1}, p_{0,j}, p_{0,j+1}$ and $\triangle p_{0,j}, p_{0,j+1}, p_{0,j+2}$. Stated another way,

$$\sum_{j=0}^{|S_0|-1} t_j \leqslant 2|S_1| \leqslant 2n.$$

Thus, the overall running time of this algorithm is

$$T(n) = O(n \log n) + \sum_{j=0}^{|S_0|-1} O(1 + t_j) = O(n \log n),$$

as claimed.

4.2.3. Removing 2 outliers

Next we consider the case $c = 2$. In this case, the optimal solution S' has one of the three following forms:

(1) S' contains two consecutive points $p_{0,j}$ and $p_{0,j+1}$ of S_0.
(2) S' contains two non-consecutive points p_{0,j_1} and p_{0,j_2} of S_0 (with $j_2 \notin \{j_1 - 1, j, j_1 + 1\}$).
(3) S' contains one point $p_{0,j}$ of S_0 and one point $p_{1,j'}$ of S_1.

The solutions of Type 1 can be found in much the same way as the algorithm for the case $c = 1$. For each $j \in \{0, \ldots, |S_0| - 1\}$ we compute the difference in area between $\text{conv}(S)$ and $\text{conv}(S \setminus \{p_{0,j}, p_{0,j+1}\})$. The analysis remains exactly the same as before except that, now, each point of S_1 can appear in at most 3 area computations, instead of only 2. Thus, all solutions of Type 1 can be evaluated in $O(n \log n)$ time.

The solutions of Type 3 can also be found in a similar manner. For each point $p_{0,j} \in S_0$ we remove $p_{0,j}$ to expose a sequence $p_{1,k}, \ldots, p_{1,\ell}$ of S_1 and compute the area of $\text{conv}(S \setminus \{p_{0,j}\})$. We then remove each of $p_{1,k}, \ldots, p_{1,\ell}$ in turn (exposing a chain of points from S_2) and compute the area of the resulting convex hull. To analyze the cost of all these, we observe that each point $p_{1,j} \in S_1$ appears in at most 2 subproblems because there are at most 2 points in S_0 whose removal causes $p_{1,j}$ to appear on the convex hull. Similarly, for each point $p_{2,j} \in S_2$ there are at most 2 points of S_1 whose removal causes $p_{2,j}$ to appear on the convex hull. Thus, each point in S_1 appears in at most 2 subproblems and each point in S_2 appears in at most 4 area computations. The overall running time of this algorithm is therefore bounded by

$$O\big(n \log n + |S_0| + 2|S_1| + 4|S_2|\big) = O(n \log n),$$

as required.

Finally, we consider solutions of Type 2. To find these we compute, for each $p_{0,j} \in S_0$, the difference x_j between the area of $\text{conv}(S \setminus \{p_{0,j}\})$ and $\text{conv}(S)$ using the technique described for the case $c = 1$. In this way, we reduce the problem to that of finding two indices $0 \leqslant j_1, j_2 < |S_0|$ with $j_2 \geqslant j_1 + 2$ such that $x_{j_1} + x_{j_2}$ is maximum. This can be accomplished by computing the quantity

$$D_{|S_0|} = \max\big\{x_{j_1} + x_{j_2} \colon\ 0 \leqslant j_1, j_2 \leqslant |S_0| \text{ and } j_2 \geqslant j_1 + 2\big\},$$

which can be computed in $O(|S_0|)$ time using the dynamic-programming recurrence

$$D_j = \max\big\{D_{j-1}, x_j + \max\{x_0, \ldots, x_{j-2}\}\big\}.$$

Since the best solution of each of the three types can be found in $O(n \log n)$ time, we can find the overall best solution in $O(n \log n)$ time by keeping the best of the three.

4.2.4. Removing c outliers

The solution for the case $c = 2$ illustrates all of the ideas used in our algorithm. We begin by enumerating the combinatorial types of solutions and then compute the best solution of each type. The algorithm for computing the best solution of each type is a divide-and-conquer algorithm whose merge step is accomplished by solving a dynamic programming problem (as in the Type 2 solutions described above).

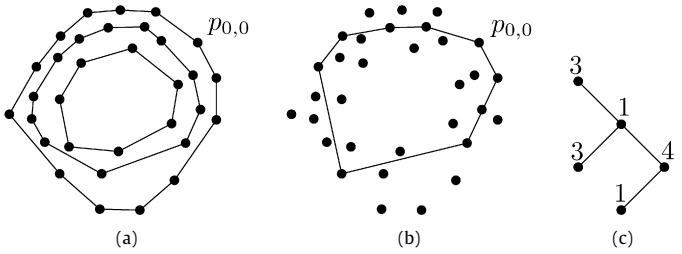

Fig. 2. Examples of (a) a point set S, (b) a solution $S \setminus S'$, and (c) the solution tree for S'.

4.2.5. The types of solutions

We represent the type of a solution as a rooted ordered binary tree in which each node is labeled with a positive integer and the sum of all node labels is c. We call such trees *solution trees* and interpret them as follows (refer to Fig. 2 for an example): Any solution removes some elements of S_0 and the elements removed come in d groups $G_0^1, \ldots, G_0^d$ of consecutive elements with each group separated by at least one element of S_0. The sizes of these groups are given by the labels of the nodes on the rightmost path in T, in the order in which they occur. That is, the jth node, N_0^j on the rightmost path of T has the label $|G_0^j|$.

For some group G_0^j, let $p_{1,k}, \ldots, p_{1,\ell}$ denote the points of S_1 that appear on the boundary of $\text{conv}(S \setminus G_0^j)$. Any solution removes some subset of $p_{1,k}, \ldots, p_{1,\ell}$ of elements from S_1. Again, this subset can be partitioned into groups of consecutive elements with any two groups separated by at least one element of S_1. In the solution tree T, the sizes of these groups are given, in the order in which they occur, by the labels of the rightmost path in the subtree of T rooted at the left child of N_0^j.

This process is repeated recursively: Let $S_{<i} = \bigcup_{j=0}^{i-1} S_j$ and let $S'_{<i} = S_{<i} \cap S'$. For each consecutive group G_i^j of nodes that are removed from S_i, let $p_{i+1,k}, \ldots, p_{i+1,\ell}$ denote the vertices on S_{i+1} that appear on the boundary of $CH(S \setminus (S'_{<i} \cup G_i^j))$. In the solution tree T, the rightmost path of the left subtree of the node representing G_i^j contains nodes representing the sizes of consecutive groups of nodes that are removed from the chain $p_{i+1,k}, \ldots, p_{i+1,l}$ of S_{i+1}. In this way, any solution S' to the outlier removal problem that does not remove both $p_{0,0}$ and $p_{0,-1}$ maps to a unique solution tree.

Since we will be exploring all possible solution trees, we require the following lemma to show that, for small values of c, there are not too many solution trees:

Lemma 8. *The number of solution trees is at most $O(C(2c))$, where $C(r) = \binom{2r}{r}/(r+1)$ is the rth Catalan number.*

Proof. Given a solution tree T, we convert it to an unlabeled rooted binary tree as follows: First, for each node N of T whose label is ℓ, we color N black, leave N's left child unchanged and replace N's right child with a (right) path of $\ell - 1$ *white* nodes, the last node of which has N's original right child as its right child. Note that this gives us a binary tree with exactly c nodes in which no white node has a left child. Next, for each black node we add a white leaf, if necessary, to guarantee that each black node has a left child. This gives us an unlabeled rooted ordered binary tree T' with at most $2c$ nodes.

Observe that, given only T', we can recover T and its labels since we can recognize white nodes by the fact that they have no left child. Thus, the number of solution trees is at most equal to the number of unlabeled rooted ordered binary trees with at most $2c$ nodes. The number of such trees is

$$\sum_{i=1}^{2c} C(i) = O\big(C(2c)\big)$$

[14], as required. □

Note that the proof of Lemma 8 shows how to convert a binary tree with at most $2c$ nodes into a solution tree. Thus, we can enumerate all possible solution trees by enumerating all binary trees having at most $2c$ nodes using, for example, the algorithm of Solomon and Finkel [21].

4.2.6. Computing the solution of a specific type

In this section we describe an algorithm that takes as input a solution tree T and outputs the value of the optimal solution S' whose solution tree is T.

The algorithm we describe is recursive and operates on a subchain $p_{i,j}, \ldots, p_{i,k}$ of S_i along with a solution (sub)tree T. The algorithm requires that some subset of $S_{<i}$ has already been removed from S so that $p_{i,j}, \ldots, p_{i,k}$ are on the boundary of the convex hull of the current point set. The algorithm finds an optimal solution of type T such that the only points

removed from the convex hull of the current point set are in $p_{i,j}, \ldots, p_{i,k}$. The initialization of this algorithm requires special care and will be described later.

Let d denote the number of nodes on the rightmost path of T. The algorithm accomplishes its task by recursively solving $O(d(k-j+1))$ subproblems on the left children of these d nodes and then combining these solutions using dynamic programming. The following pseudocode gives a detailed description of the algorithm's operation with the exception of the dynamic programming component, whose description and analysis is discussed in the next subsection. Note that the algorithm below only computes the maximum amount of area (perimeter) that can be removed from conv(S) by a solution S' whose solution tree is T. To obtain the points in S' the algorithm can be augmented using the standard trick of remembering, whenever the algorithm takes the maximum of two values, which of the two values produced the maximum. The details of this are standard and omitted here.

```
FindOptimalOfType(T, i, j, k)
 1: if T is empty then
 2:    return 0
 3: d ← the number of nodes on the rightmost path of T
 4: N ← the sum of node labels on rightmost path of T
 5: if k − j < N + d then
 6:    return −∞ {* No solution possible *}
 7: for g = 1 to d do
 8:    N_g ← gth node on the rightmost path of T
 9:    c_g = label(N_g)
10:    for ℓ = j to k − c_g + 1 do
11:       delete S_{i,ℓ}, ..., S_{i,ℓ+c_g−1} from S_i exposing S_{i+1,j'}, ..., S_{i+1,k'} on S_{i+1}
12:       s ← reduction in area (perimeter) obtained by the deletion of S_{i,ℓ}, ..., S_{i,ℓ+c_g−1}
13:       X_{g,ℓ−j+1} ← s + FindOptimalOfType(left(N_g), i + 1, j', k')
14:       reinsert S_{i,ℓ}, ..., S_{i,ℓ+c_g−1} into S_i
15: return CombineSolutions(X, d, k − j + 1, c_1, ..., c_d)
```

The call to CombineSolutions in the last line of the algorithm is a dynamic programming subroutine described in the next section that runs in $O(d(k-g))$ time. The CombineSolutions subroutine computes the optimal locations of the groups of points represented by $N_1, \ldots, N_d$ in T. At the topmost level, the algorithm is called as FindOptimalOfType($T, 0, 0, |S_0| - 1$).

To analyze the cost of FindOptimalOfType it suffices to determine, for each point $p_{i,j}$, the maximum number of times $p_{i,j}$ is deleted (in line 9) by the algorithm. All other work done by the algorithm can be bounded in terms of this quantity. For points $p_{0,j} \in S_0$, each point is deleted exactly g_0, times, where g_0 is the sum of labels of nodes on the rightmost path of T.

More generally, let g_i denote the sum of labels of all nodes N of T for which the path from the root of T to N makes exactly i left turns. (These nodes correspond to groups that are deleted from S_i.) Consider some point $p_{i,j} \in S_i$, for $i \geqslant 1$. By Carathéodory's Theorem [10], each such point is contained in some triangle $\Delta_{i,j} = \triangle p_{i-1,\ell_1}, p_{i-1,\ell_2}, p_{i-1,\ell_3}$. If $p_{i,j}$ is deleted by FindOptimalOfType, then it is on the boundary of the convex hull of the current point set. However, this implies that at least one of the three vertices of $\Delta_{i,j}$ must be deleted from the current point set. Thus, if we define m_i as the maximum number of times a point of S_i is deleted by FindOptimalOfType then we have the relationships:

$$m_i \leqslant \begin{cases} g_0 & \text{if } i = 0, \\ 3m_{i-1}g_i & \text{if } 1 \leqslant i < c, \\ 0 & \text{otherwise.} \end{cases}$$

Using the fact that $g_i \leqslant c$, we obtain the (extremely loose) upper bound $m_i \leqslant (3c)^{i+1}$. This implies that the points of S_c (which are never deleted) appear in at most $3m_{c-1}$ subproblems.

All that remains is to describe how we initialize the algorithm. To do this, we first check if the label at the root of T is equal to $|S_0|$. If so, and the root of T has a right child, then we immediately return the value $-\infty$ since no solution is possible. Otherwise (the root of T has no right child) we remove S_0 from our point set and recurse on $S \setminus S_0$ and the left child of the root of T.

Otherwise (the label at the root of T is not equal to $|S_0|$) then we make $c+1$ calls to FindOptimalOfType using as inputs the chains $p_{0,t}, \ldots, p_{0,|S_0|-2+t}$, for $t = 0, \ldots, c$. Note that each such call fixes one of $p_{0,0}, \ldots, p_{0,c}$. Since the optimal solution must leave one of these $c+1$ points we are guaranteed that at least one of these calls will find the optimal solution.

Putting all this together we obtain:

Lemma 9. *The algorithm* FindOptimalOfType *finds the optimal solution whose solution tree is T in $O((3c)^{c+1}n)$ time.*

4.2.7. Combining the solutions

One aspect of the algorithm that we have not yet described is how the subroutine CombineSolutions works. This subroutine is given positive integers $c_1, \ldots, c_d$ and a $d \times m$ positive real-valued matrix X and must find indices $1 \leqslant i_1, \ldots, i_d \leqslant m - c_d$ such that

$$i_{j+1} \geqslant i_j + c_j + 1$$

for all $1 \leqslant j < d$ and such that the sum

$$h(i_1, \ldots, i_d) = \sum_{j=1}^{d} X_{j,i_j}$$

is maximum. In the terminology of the previous section, the value d is the number of nodes in the rightmost path of T, $m = k - j + 1$, and the indices $i_1, \ldots, i_d$ correspond to the indices of the first element of each group on the chain $p_{i,j}, \ldots, p_{i,k}$ considered by the algorithm. We solve this problem by filling out the $d \times m$ dynamic programming table:

$$D_{j,\ell} = \max\bigl\{h(i_1, \ldots, i_j)\colon\ 1 \leqslant i_1, \ldots, i_j \leqslant \ell,\ \text{and } i_{j'+1} \geqslant i_{j'} + c_{j'} + 1 \text{ for all } 1 \leqslant j' < j\bigr\}$$

for $j = 1, \ldots, d$ and $\ell = 1, \ldots, m - c_j + 1$. We can do this in $O(dm)$ time because the table entries satisfy the recurrence

$$D_{j,\ell} = \max\{D_{j,\ell-1}, D_{j-1,\ell-c_{j-1}-1} + X_{j,\ell}\}$$

where we use the convention that $D_{j,\ell} = 0$ if $j = 0$ and $D_{j,\ell} = -\infty$ if $\ell < 0$. This yields the last lemma required by the algorithm:

Lemma 10. *The function* CombineSolutions *can be implemented in* $O(dm)$ *time.*

This (finally) completes the proof of:

Theorem 11. *There exists an* $O\bigl(n \log n + \binom{4c}{2c}(3c)^{c+1} n\bigr)$ *time algorithm to solve the* (*convex hull*) *area/perimeter-based* c *outlier removal problem.*

5. Conclusions

We have given algorithms for removing c outliers to optimize various criteria. For any constant value of c, all our algorithms run in $O(n \log n)$ time. It is interesting to note that our simplest and fastest algorithms, running in $O(n)$ time, are for methods based on bounding boxes, which are not invariant to rotations of the input. At the next level is the diameter-based method, which is invariant to rotations of the input, but not invariant to non-uniform scaling. The most complicated of our algorithms is the algorithm based on convex hull area, whose results are invariant to any affine transformations of the input.

We conclude with a few directions for further research:

(1) Our algorithm for minimizing the diameter of $S \setminus S'$ extends readily to point sets S in $\mathbb{R}^3$. Using an $O(n \log n)$ diameter-finding algorithm [7,18] and the fastest algorithms for Vertex-Cover [6] we obtain an algorithm with running time $O(cn \log n + c^4 1.27^{(c^2)})$. Is there an algorithm whose running time depends only polynomially on c?
(2) The running time of our algorithm for minimizing the area/perimeter of the convex hull has a superpolynomial dependence on the value of c. Does there exist an algorithm that is polynomial in c but that still runs in $O(n \log n)$ time for any fixed value of c?

References

[1] A. Aggarwal, H. Imai, N. Katoh, S. Suri, Finding k points with minimum diameter and related problems, Journal of Algorithms 12 (1991) 38–56.
[2] R. Atanassov, P. Morin, S. Wuhrer, Removing outliers to minimize area and perimeter, in: Proceedings of the 18th Canadian Conference on Computational Geometry (CCCG 2006) 2006.
[3] M. Ben-Or, Lower bounds for algebraic computation trees (preliminary report), in: Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC'83), 1983, pp. 80–86.
[4] T.M. Chan, Low-dimensional linear programming with violations, SIAM Journal on Computing 32 (2005) 879–893.
[5] B. Chazelle, On the convex layers of a planar set, IEEE Transactions on Information Theory 31 (1985) 509–517.
[6] J. Chen, I.A. Kanj, W. Jia, Vertex cover: Further observations and further improvements, Journal of Algorithms 41 (2) (2001) 280–301.
[7] K.L. Clarkson, P.W. Shor, Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental, in: Proceedings of the Fourth Annual ACM Symposium on Computational Geometry (SoCG'88), 1988, pp. 12–17.
[8] D. Dobkin, R. Drysdale, L. Guibas, Finding smallest polygons, Computational Geometry: Theory and Applications 1 (1983) 181–214.
[9] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, 1999.
[10] J. Eckhoff, Helly, Radon, and Carathéodory type theorems, in: P.M. Gruber, J.M. Wills (Eds.), Handbook of Convex Geometry, vol. B, North-Holland, 1993, pp. 389–448, Chapter 2.1.
[11] D. Eppstein, New algorithms for minimum area k-gons, in: Proceedings of the Third ACM–SIAM Symposium on Discrete Algorithms (SODA 1992), 1992.

[12] D. Eppstein, J. Erickson, Iterated nearest neighbors and finding minimal polytopes, Discrete & Computational Geometry 11 (1994) 321–350.
[13] D. Eppstein, M. Overmars, G. Rote, G. Woeginger, Finding minimum area k-gons, Discrete & Computational Geometry 7 (1992) 45–58.
[14] R.L. Graham, D.E. Knuth, O. Patashnik, Concrete Mathematics: A Foundation for Computer Science, second ed., Addison-Wesley, 1994.
[15] J. Hershberger, S. Suri, Applications of a semi-dynamic convex hull algorithm, BIT 32 (2) (1992) 249–267.
[16] J. Matoušek, On geometric optimization with few violated constraints, Discrete & Computational Geometry 14 (1995) 365–384.
[17] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, Springer-Verlag, New York, 1985.
[18] E.A. Ramos, An optimal deterministic algorithm for computing the diameter of a three-dimensional point set, Discrete & Computational Geometry 26 (2) (2001) 233–244.
[19] M. Segal, K. Kedem, Enclosing k points in the smallest axis parallel rectangle, Information Processing Letters 65 (2) (1998) 95–99.
[20] M.I. Shamos, Computational geometry, PhD thesis, Yale University, 1978.
[21] M. Solomon, R.A. Finkel, A note on enumerating binary trees, Journal of the ACM 27 (1) (1980) 3–5.